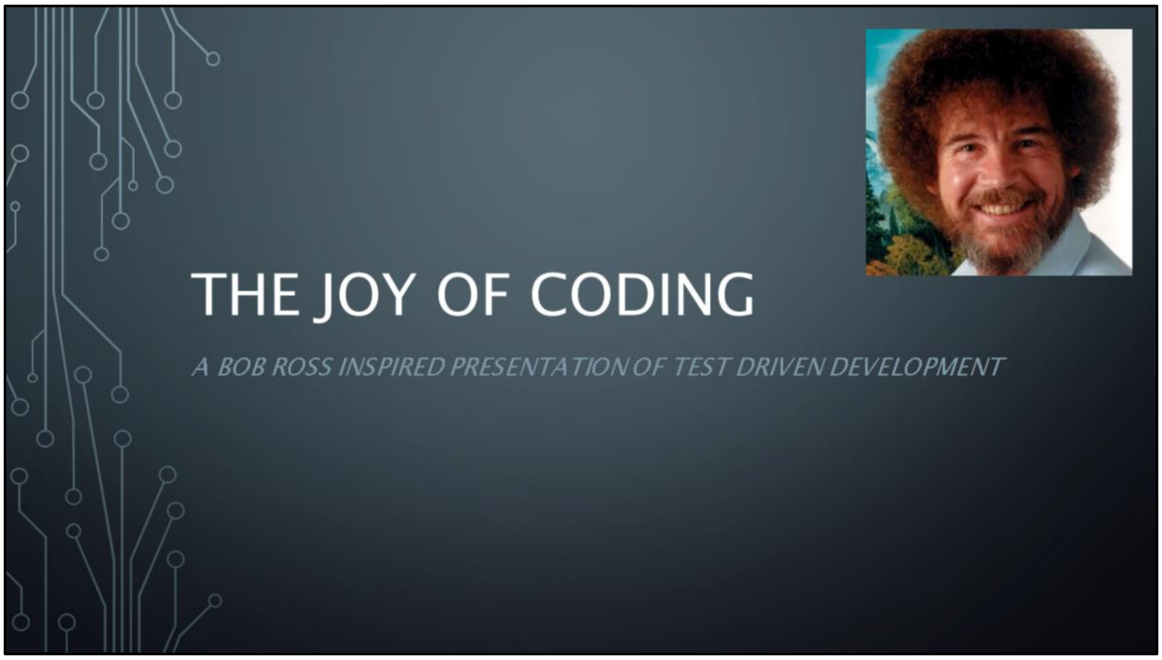
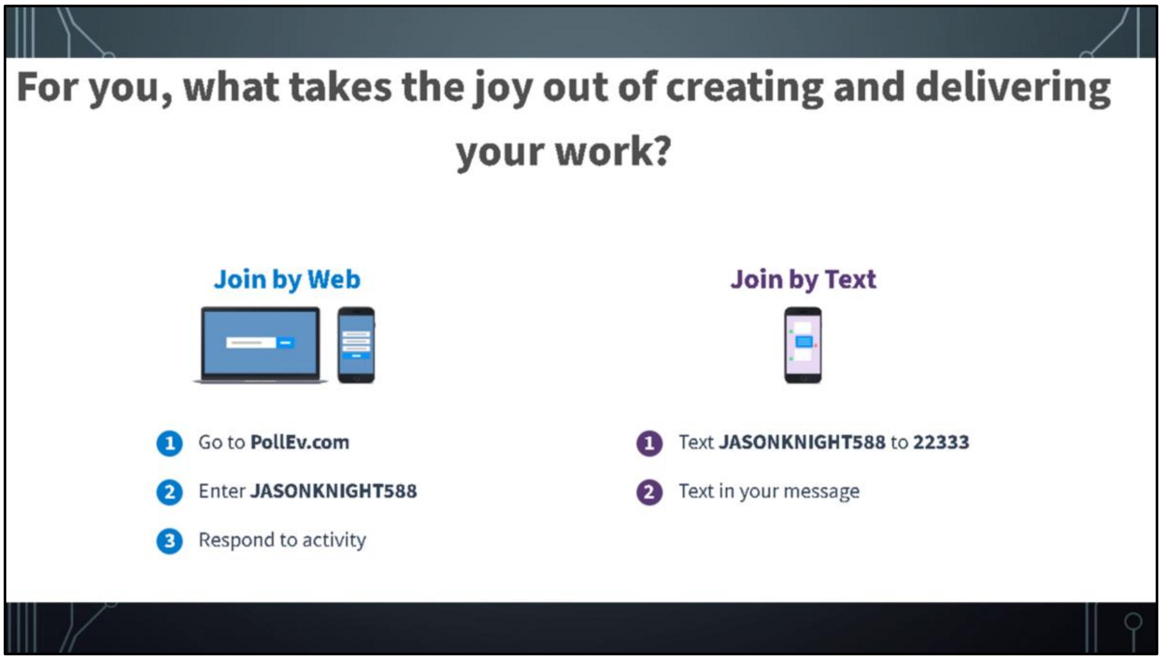


# THE JOY OF CODING

*A BOB ROSS INSPIRED PRESENTATION OF TEST DRIVEN DEVELOPMENT*

Bob Ross entertained and educated thousands through his program, “The Joy of Painting.” The formula was simple: demonstrate how rewarding painting can be, equip viewers with practical tips, and encourage them to give it a try for themselves. My intention is to follow in his footsteps. I’d like to invite you to learn unit testing through a design practice called *Test-Driven Development (TDD)*. I hope to show you how rewarding this practice can be and how anyone with a little inspiration and intention can design wonderful features that work just as you want them to. What a joyful thing! I’ll show you practical tips and encourage you to give it a try for yourself. If you’re not a coder, this will still be worth your time. You’ll see how, if given the right support and equipment, the developers in your project can produce high-quality and help you keep your promised to stakeholders and users.

To best appreciate the Joy of Coding, let’s start by asking, “What takes the joy out of creating and delivering our work?”



## For you, what takes the joy out of creating and delivering your work?

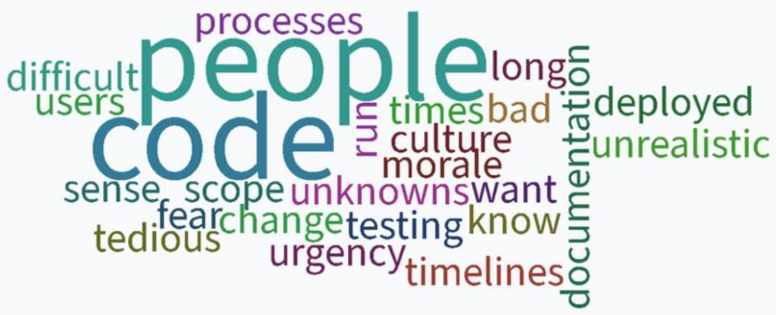
**Join by Web**

1. Go to **PollEv.com**
2. Enter **JASONKNIGHT588**
3. Respond to activity

**Join by Text**

1. Text **JASONKNIGHT588** to **22333**
2. Text in your message

## For you, what takes the joy out of creating and delivering your work?



No One likes bugs. Full Stop.

Developers don't like shipping code that doesn't work
Project Managers don't like that bugs can unpredictably delay schedules
Product Owners want to make raving fans of their users and bugs can destroy good will

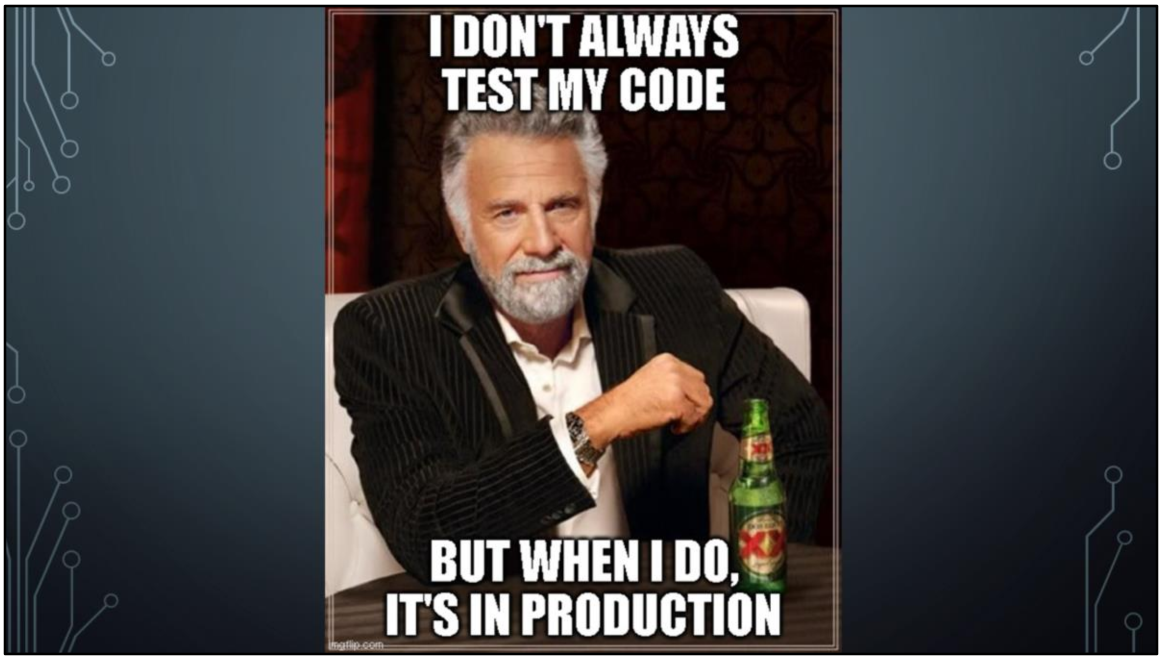
Bugs often come from code that is not well tested and from architecture that is poorly designed. If the code we write isn't fully tested, developers know that our software and our users will find a way to expose it's weaknesses.

Most importantly, bugs are waste. The time you spend fixing a bug is time you could be using to produce working, valuable software.

Regarding testing, please know I'm not throwing QA under the bus here. Too often, poorly designed and implemented code is dumped on QA folks to "verify" with manual tests. This is unfair, as the QA folks generally test by running the application through select scenarios. "Full regression" is chosen only rarely and still may not catch everything. Therefore, manual tests performed in say a Scrum sprint can amount to a sort of “due diligence” activity performed at the last minute and has the effect of stressing out QA folks towards the end of a Sprint.

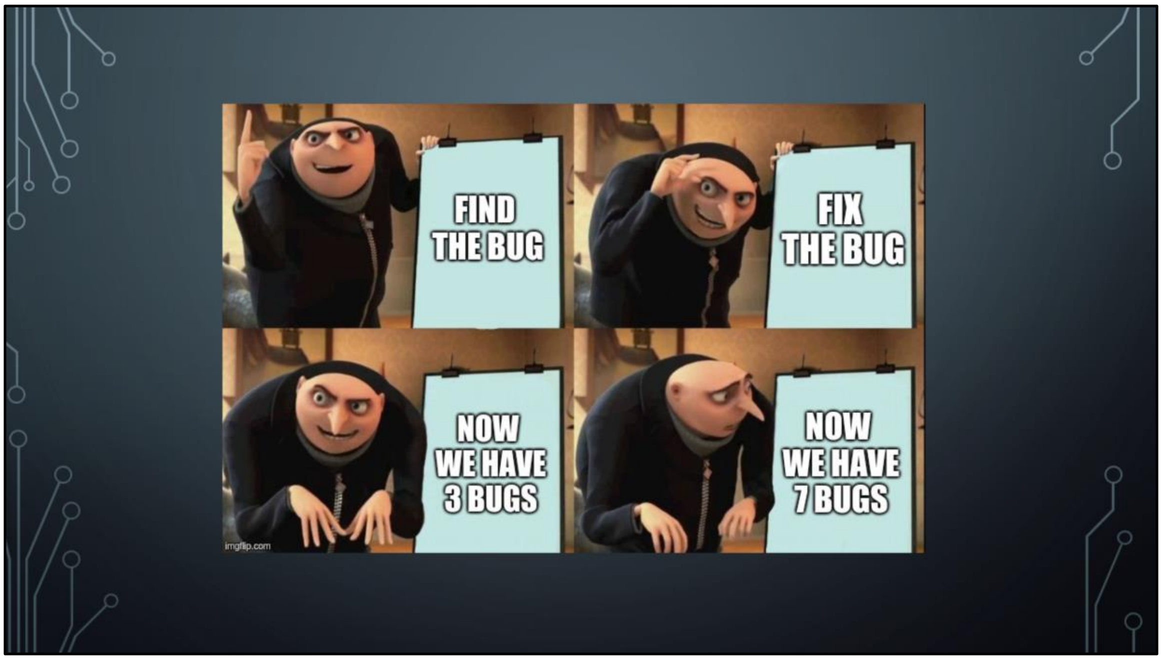
Here are just a few ways bugs have negatively impacted my life.

We don’t get late night and weekend calls about  well-built software. No, we get those calls because of bugs and bad architecture. This is an oversimplification, but in my experience they account for a large amount of after hours stress.

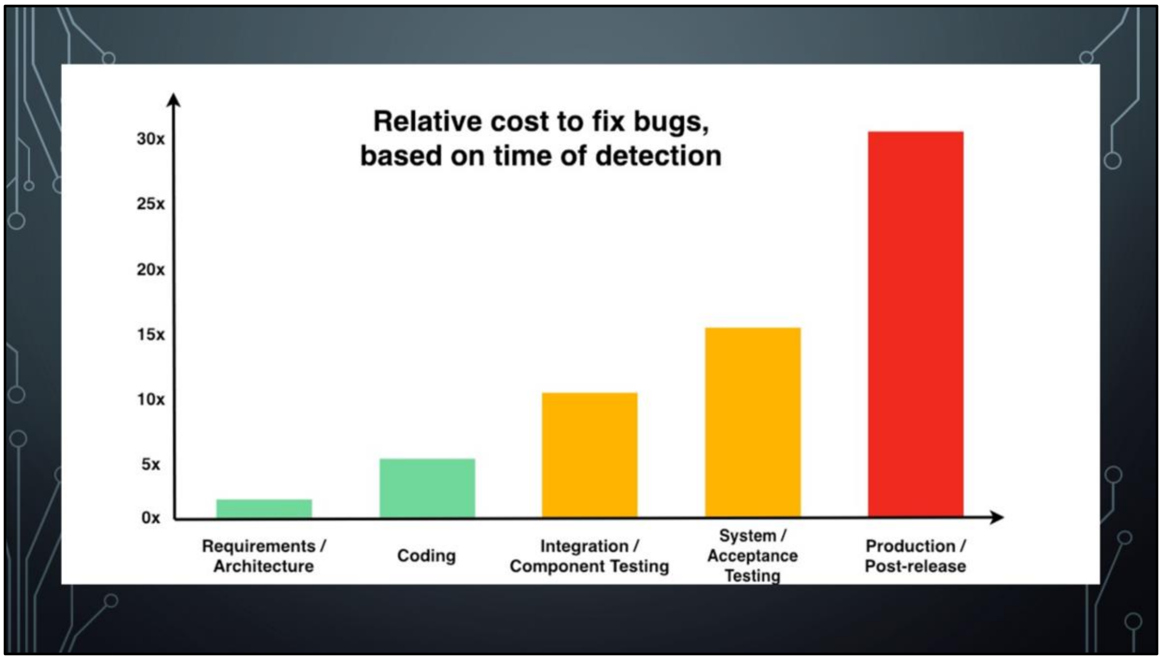

I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO, IT'S IN PRODUCTION
imgflip.com

We get those late night calls because (again, an oversimplification) something is not working in production that requires an emergency fix. At this point, the testing that could of and arguably *should have* happened in the Sprint is performed.

Fixing bugs is easier said than done. Everyone involved in delivering software has probably experienced the “phantom bug.” It shows up then it disappears. Developers may even push a fix which *appears* to exercise the demon from the codebase only to have it reappear inexplicably later. Users are getting angrier and you’re already sleep deprived.

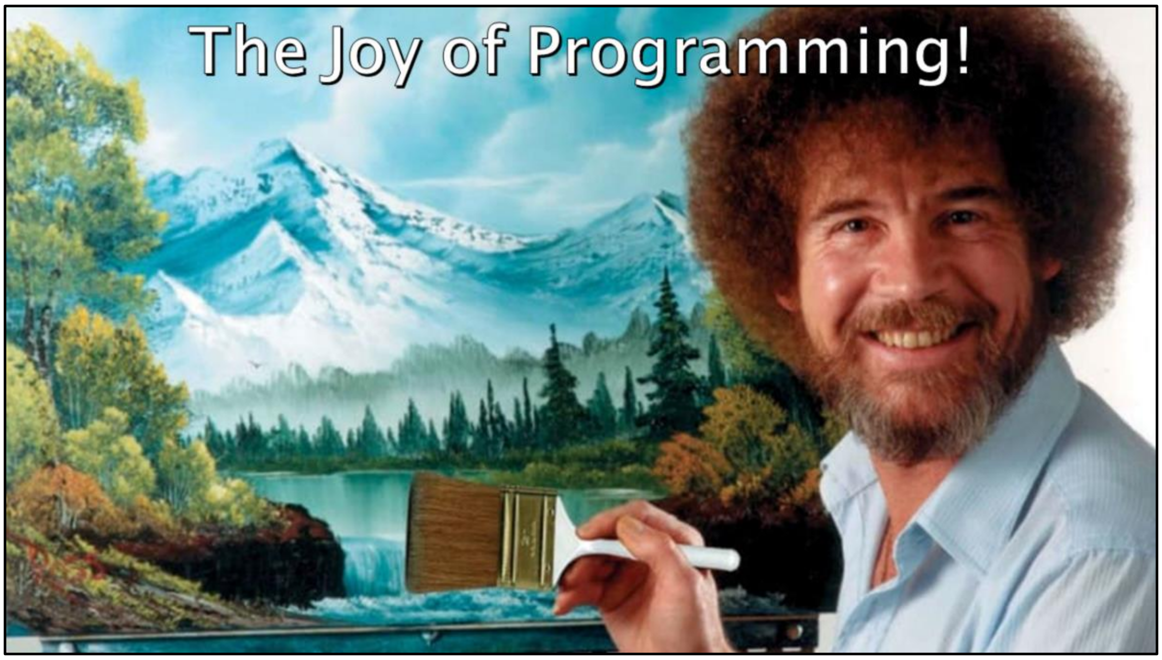


In particularly gnarly software, you may have additional difficulty. Overly complicated architecture and untested code means that what you think is a fix may actually break other working code. If all you have are manual tests to run, you'll run into the full vs partial regression conundrum; there's not time for a full regression and you're not sure which subset of tests to conduct to prevent new bugs from resulting from the change.



I’m focusing on bugs a great deal in this introduction because they represent significant waste to the business and sap the joy out of the work. The National Institute of Science and technology have gathered enough data to estimate the cost of bugs to an enterprise. Furthermore, they show that the relative cost to fix them rises quickly the later they are detected in the development process.

The implications of this graph AND the previous slides demonstrate the incentive we all share to find a better way to do this. You might say I’ve described the Agony of Programming. Next, I’d like to show you…



The Joy of Programming! I'm going to show you how to create a simple application using a technique called Test-Driven Development. I invite you to come with me on this journey where we'll take a blank canvas and turn it into a well-formed work of art. It will be fully tested, do exactly what we want it to and even yield a surprise at the end!

If you are new to testing, I hope you will see how easily you can get started learning to write tests.

If you are a project manager, I hope you will begin to imagine how much more pleasant it can be to work with high-quality software written using TDD and automated testing.

For Product Owners, I hope you will envision the value your teams will be able to deliver from a codebase this stable and open to being changed.

If you have been writing automated tests for ages, I hope you will find the following of interest All the code you'll see me write will be done without touching my mouse. It will all be produced from my keyboard alone.

Without further ado, on with the show!